Synthèse de l'interconnexion des mémoires dans un contexte de système intégré multi-processeurs

Lahcene Abdelouel, Daniel Chillet, Olivier Sentieys

Synthèse de l'interconnexion des mémoires dans un contexte de système intégré multiprocesseurs

Lahcene Abdelouel[†], **Daniel Chillet**[‡], **Olivier Sentieys**[‡]
[†] ENP Alger, 10, Avenue Hassen Badi El Harrach, Alger, ALGERIE
[‡] R2D2 - IRISA - ENSSAT - Université de Rennes 1, 6, rue de kerampont, 22300 Lannion, FRANCE
prenom.nom@irisa.fr

Résumé : Le développement important des systèmes intégrés sur puce (SoC)[1] montre que ces systèmes embarquent une part de plus en plus important de mémoires. Globalement, les SoCs sont constitués de cœurs de traitement généraliste et/ou spécifique et accèdent, via un réseau sur silicium (NoC[2]), à une organisation mémoire plus ou moins complexe. La connexion de cette organisation mémoire aux différents acteurs du système pour l'acheminement des données depuis ou vers la mémoire est un point critique du système. Dans cet article, nous nous intéressons à ce problème d'interconnexion entre les acteurs et la mémoire pour une application temps réel à séquence d'accès déterministe. L'approche que nous proposons, consiste en la mise en place d'une interface flexible entre les mémoires et le réseau véhiculant l'ensemble des requêtes des acteurs du système. Cette interface doit s'assurer que le système est bien en mesure de sauvegarder et de récupérer des données dans les mémoires. Nous développons une formulation ILP (Integer Linear Programming) de ce problème et nous présentons un exemple de mise en œuvre d'application de traitement d'images (DCT[3]) pour illustrer l'échange des données et le rôle de l'interface.

Mots-clés : System On Chip (SoC), Hiérarchie mémoire, Interconnexion, Architecture

1 INTRODUCTION

La densité d'intégration des transistors sur le silicium offre, aujourd'hui, aux concepteurs de systèmes la capacité de concevoir une seule puce pour y intégrer la totalité de leurs applications. Ces systèmes (SoC) sont constitués généralement de cœur(s) de processeur, d'accélérateur(s) matériel(s), d'interface(s) avec le monde extérieur (convertisseur analogique numérique et *vice versa*), de réseau(x) et de mémoire(s). Bien que la complexité des applications à implémenter dans ces circuits soit de plus en plus importante, les prédictions d'évolution nous annoncent que la mémoire est la partie qui "occupera" la plus importante surface du circuit, et qu'elle sera par conséquent la plus grande "consommatrice" d'énergie. En effet, les prédictions prévoient que la mémoire s'étendra sur plus de 90% de la surface à l'aube de 2014 (voir figure 1). Les 10% restant étant alors alloués à de la logique classique et/ou à des parties matérielles très spécifique.

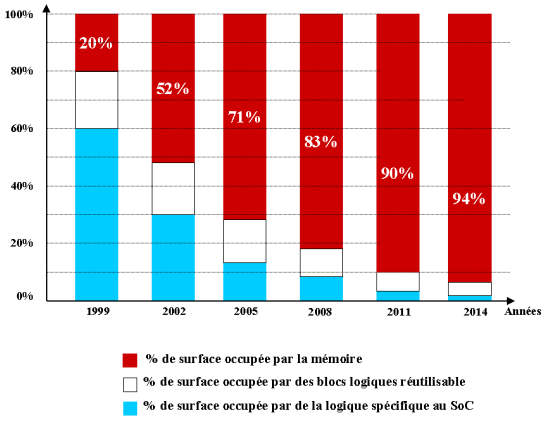

FIG. 1 – Part de la mémoire dans les SoC, source SIA Roadmap

Cette évolution sans cesse croissante est essentiellement due à l'évolution des applications multimédia (audio, vidéo et images) qui manipulent des volumes de données très importants. Cela se traduit finalement par un très grand nombre de transferts mémoires pour satisfaire les besoins des différentes tâches de l'application. D'un point de vue énergétique, ces transferts sont une source de consommation très importante, et de ce point de vue, il est inconcevable d'envisager un système dans lequel les données de l'application seraient perpétuellement déplacées en mémoire pour satisfaire les ressources de calcul du système (exécutant les tâches de l'application). Le problème difficile de l'interconnexion apparaît alors comme un point où les contraintes dues aux SOC interviennent fortement.

Les travaux que nous présentons dans ce papier visent à proposer une solution d'interconnexions minimale sous contrainte de satisfaction des transferts de données (d'une applications temps réel à séquence d'accès déterministe) vers les ressources de calcul du système. L'organisation du papier suit la découpe suivante. La section 2 présente un état de l'art du domaine. La section 3 présente notre méthodologie et une formalisation du problème de l'in-

[1] SoC : System on Chip
[2] NoC :Network on Chip
[3] DCT : Discret Cosinus Transform

terconnexion du système mémoire avec les ressources de calcul du système. Un exemple illustre nos propos à la section 4. Enfin, nous concluons et donnons quelques perspectives.

2 ETAT DE L'ART

De nombreuses études ont abordé la problématique liée à la mémorisation au sein des systèmes de type SoC. Les approches proposées s'intéressent notamment à la mise en place d'une structure mémoire hiérarchique, à l'allocation de l'espace d'adressage, à la gestion des conflits d'accès à une seule mémoire, ou encore aux problèmes de la consommation dans les circuits. Nous donnons quelques éléments de ces travaux dans les paragraphes suivants.

2.1 Etudes concernant la mise en place de hiérarchie mémoire

A l'instar des systèmes microprocesseurs, la mise en place de structures mémoires hiérarchiques pour répondre aux besoins en performances des systèmes intégrés sur puce est une solution très largement explorée. Les travaux de recherche dans ce domaine peuvent être classés en trois catégories, selon l'approche suivie pour aborder le problème :

– La première approche consiste à déterminer, pour une application donnée, la hiérarchie optimale (nombre de niveaux, taille des mémoires...) en fonction de la localité des données [Jacob *et al.*, 1996] ;
– La seconde approche consiste à trouver, pour une architecture cible prédéfinie et une hiérarchie mémoire fixe, le placement optimal de chaque donnée dans le niveau de la hiérarchie le plus adapté, afin de répondre aux contraintes de l'application [Crummey *et al.*, 2001, Ouaiss et Vemuri, 2001, Kavvadias *et al.*, 2001] ;
– Enfin, la troisième approche combine la recherche d'une hiérarchie avec l'évaluation du placement des données dans le niveau hiérarchique le plus adapté et cela sous contrainte fixée par l'application [Catthoor *et al.*, 1998, Catthoor, 1999].

La première approche est une approche matérielle et elle convient mieux à la conception des circuits spécifiques de type ASIC. La seconde approche peut être considérée comme une approche logicielle, puisque l'architecture mémoire est fixée d'avance. Tous les travaux sur la gestion des caches dans les systèmes classiques ou sur les optimisations au niveau code, peuvent être classés dans cette catégorie. Pour les systèmes intégrés sur puce, la dernière approche, basée sur une démarche conjointe logicielle/matérielle semble la mieux adaptée. C'est de loin, l'approche la plus complète, mais la complexité d'automatisation la rend difficilement utilisable [Catthoor *et al.*, 2000].

2.2 Problème de gestion de l'espace d'adressage

Lorsque les applications manipulent des tableaux de données, une phase importante de la conception consiste à allouer l'espace mémoire pour ces structures de données. La technique classique consiste à simplement affecter un offset à chaque tableau de données, et lors de l'accès à un élément, il suffit alors de réaliser une opération d'addition entre l'indice de l'élément recherché et l'offset en question. Cette technique simple présente l'inconvénient de nécessiter un opérateur matériel pour la réalisation de l'opération d'addition, et dans certains cas cela peut être pénalisant du point de vue du temps d'accès à la données. Dans [Schmit et Thomas, 1998], les auteurs présentent trois techniques qui permettent de limiter le coût matériel ainsi que temporel lié à cette opération (Le principe est illustré figure 2.a).

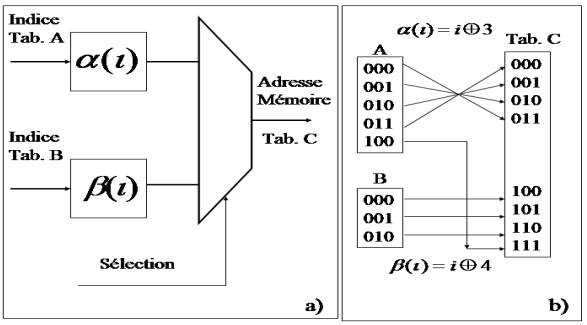

FIG. 2 – Illustration de la techniques d'allocation selon [Schmit et Thomas, 1998]

La première technique consiste à imbriquer deux tableaux de dimensions très proches, le premier tableau est positionné aux adresses paires alors que le second est positionné aux adresses impaires. La seconde technique consiste à remplacer l'opération d'addition par une opération de OU exclusif, XOR, dont le coût est moindre (voir exemple figure 2.b). Cela n'est possible sous certaines conditions de taille des tableaux. Enfin, la dernière technique proposée s'appuie sur une opération de rotation sur les bits d'adresses. Deux tableaux peuvent être imbriqués dans un espace d'adresses en plaçant, pour le premier tableau, d'un côté les éléments paires et d'un autre côté les éléments impaires, pour le second tableau le même placement peut être réalisé en prenant soin de réaliser préalablement une inversion des indices (bit reverse) des éléments du tableau.

2.3 Gestion des conflits d'accès à la mémoire

Dans [Schwaderer, 2002], les auteurs s'intéressent aux problèmes posés par l'apparition de nombreuses requêtes mémoires venant d'initiateurs différents dans un SoC.
Le contexte de l'étude est un système SoC pour lequel la mémoire est externe et qui est composé de plusieurs acteurs sollicitant la mémoire. La problématique est liée à la gestion des conflits créés par les multiples requêtes qui vont parvenir au système mémoire. En effet, l'ensemble des acteurs (blocs IP) n'ont pas forcément les mêmes cadences de fonctionnement et n'ont pas tous la même "criticité".
L'idée proposée dans cet article consiste à placer un bloc d'interface qui va récolter l'ensemble des requêtes pour

la mémoire et ordonnancer ces demandes afin de soumettre à la mémoire Off-Chip une seule séquence d'accès (un critère de qualité de service est pris en compte au moment de l'ordonnancement). Le bloc d'interface proposé est composé d'un ordonnanceur de requêtes vers la mémoire capable de gérer une interface OCP multithread. La principale critique que l'on peut formuler sur ces travaux est l'absence de mémoire interne au circuit. En effet, ce type de solution est assez éloigné des architectures des circuits actuels.

2.4 Prise en compte de l'aspect consommation

Dans l'article [Lyuh et Kim, 2004], les auteurs proposent une méthode permettant de placer les différents accès aux différentes mémoires de telle sorte qu'il soit possible de compacter le maximum d'activité sur quelques mémoires et donc faire apparaître des mémoires avec peu d'accès. Dans ce cas, les mémoires ayant peu d'accès peuvent alors être mise en veille pendant des périodes non négligeables, ce qui permet un gain en consommation relativement important. La méthode proposée part du principe que les accès ont déjà été ordonnancés pour satisfaire la partie calcul. La méthode considère l'ensemble des données comme étant scalaires et travaille sur un système composé de 2 bancs mémoires. Le gain obtenu est de l'ordre de 16%.
Ces travaux confirment l'importance du critère consommation pour la partie mémorisation des données. Notre approche pourrait, assez facilement, être enrichie par cette proposition de mise en veille des mémoires.

L'ensemble des travaux énumérés dans les paragraphes précédents présente des points intéressants, mais ne propose pas de méthode globale pour tenter de répondre efficacement au problème des interconnexions des mémoires dans un système intégré multiprocesseur. En tout état de cause, la présence de mémoires caches favorise le temps moyen d'exécution mais ne convient pas aux applications temps réel. Notre approche du problème consiste à exploiter au maximum toute la connaissance a priori de la structure de la hiérarchie mémoire et des contraintes de l'application pour mettre en place une interface mémoire flexible et reconfigurable supportant les transferts de données vers les unités de calcul.

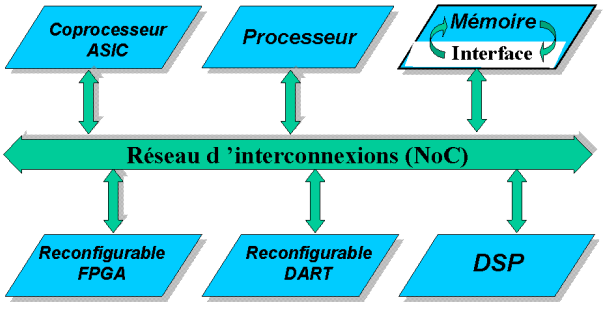

FIG. 3 – Modèle général du SoC & position de nos travaux

3 MÉTHODOLOGIE GLOBALE

Un système intégré multiprocesseurs est globalement constitué de plusieurs acteurs, chaque acteur est potentiellement demandeur d'espace de stockage (voir figure 3). De plus, ces acteurs sont susceptibles de s'échanger des données au travers de la mémoire. On peut imaginer, par exemple, une tâche (qui est acteur du système) produisant des données, ces données étant ensuite consommées par une autre tâche. Les mémoires ne pouvant, à tout instant, être connectées à tous les initiateurs de requêtes mémoires, il est nécessaire de mettre en place une interface dont le rôle est la prise en charge de la connexion des acteurs du système avec leurs cibles mémoire à l'instant t.
Dans le contexte de l'implémentation d'une application temps réel sur un SoC, le déterminisme complet de la séquence des accès aux données, permet d'envisager la construction d'une organisation mémoire spécifique en adéquation avec l'application. Pour cela, il est impératif de résoudre les deux problèmes suivants :

– Quel est, du point de vue des critères de surface, consommation et performances, le placement "optimal" des données dans les différentes mémoires du SoC ?
– Comment interconnecter ce système mémoire au reste du système de façon à minimiser le nombre de liens à mettre en place entre chaque mémoire et les autres acteurs du système ?

Les deux sections suivantes proposent une réponse aux deux questions précédentes.

3.1 Organisation Mémoire

La recherche de l'organisation mémoire "optimale" en terme de placement de données s'appuie sur la méthodologie que nous avons proposée dans [Chillet *et al.*, 2005].
L'approche développée est basée sur un modèle mémoire générique et flexible, constitué de trois niveaux hiérarchiques : deux pseudo caches de niveau 1 et 2 et une mémoire principale de niveau 3 (voir figure 4).

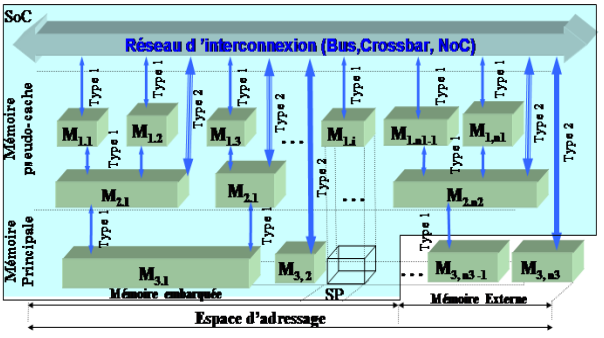

FIG. 4 – Modèle mémoire

Le flot de conception associé (figure 5) nous permet de spécifier le stockage des données dans les différentes mémoires du SoC en partant de la liste des accès mémoire définissant l'ordre des opérations et les dépendances des

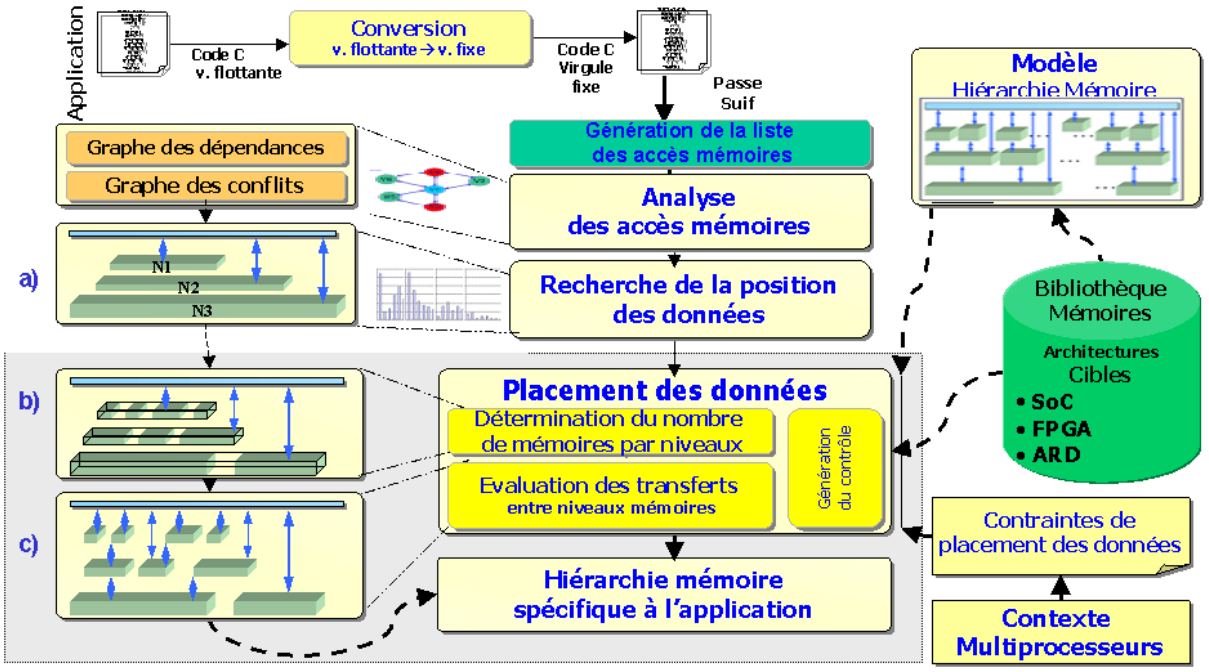

FIG. 5 – Méthodologie proposée selon une découpe en trois étapes, permettant d'aboutir à une organisation mémoire spécifique à l'application

données.

La méthodologie se déroule en trois étapes :

– la première étape a pour objectif de déterminer la position des données dans les différents niveaux de mémoire (figure 5.a) ;
– la seconde étape détermine, pour chaque niveau mémoire, le nombre de mémoires nécessaires (figure 5.b) ;
– enfin, la troisième étape évalue les transferts entre niveaux mémoires (figure 5.c).

Les deux premières étapes de cette méthodologie sont basées sur des techniques d'allocation et de distribution de données dans les mémoires et s'appuient généralement sur des algorithmes de coloriage de graphe de conflit d'accès. Dans la troisième étape, les caractéristiques de la cible matérielle vont être prises en compte de façon plus complète pour évaluer toutes les opportunités et prendre en considération les contraintes de l'architecture mémoire.

3.2 Interface des interconnexions mémoires

La troisième étape de la méthodologie exposée dans la section 3.1 nous permet de définir les connexions entre les différents blocs de l'organisation mémoire du SoC. Toutefois, dans cette première approche, les interactions de la mémoire avec les autres modules du système ne sont pas prises en compte. Cela revient à considérer que les requêtes mémoires proviennent d'une seule unité de calcul. Mais dans un contexte multiprocesseurs, les séquences de production et de consommation de données peuvent être différentes. Une réorganisation des données en mémoires peut s'avérer nécessaire pour éviter les copies de données entre mémoires. Cette opération implique deux conséquences :

– La remise en cause du placement initial des données effectué dans le cas de système mono-processeur ;
– La mise en place d'une interface d'interconnexions mémoire flexible capable de gérer toutes les requêtes d'accès aux données émanant des différents modules du SoC.

Dans la sous-section suivante, nous proposons une formulation du problème de la mise en place de cette interface sous la forme d'un problème de programmation linéaire en nombres entiers (PLNE ou ILP[4]). La PLNE est une technique classique dans la synthèse de haut niveau et constitue un cadre général particulièrement utile pour modéliser, et bien souvent résoudre, des problèmes d'optimisation dans des domaines d'application extrêmement variés. Le principe est simple : une fonction objectif linéaire à maximiser ou à minimiser, sous des contraintes linéaires et où les variables sont entières.

3.2.1 Formulation ILP du problème

En se basant sur le modèle architectural de la hiérarchie mémoire et en tenant compte du nombre de blocs mémoires nécessaire, imposé par les conflits d'accès des données (accès simultanés des données), le problème à résoudre est le suivant.

Comment déterminer le placement des données dans les bancs mémoires de façon à minimiser le nombre de liens à mettre en place entre chaque mémoire et les acteurs qui produisent et/ou consomment des données ?

[4] ILP :*Integer Linear Programming*

En exploitant au maximum toute les connaissances de la structure de la hiérarchie mémoire et des contraintes du problème, nous donnons ci-dessous, une expression sous forme ILP de ce problème.

- Soit $Data$ l'ensemble des données de l'algorithme :
  $Data = \{D_i\}_{\forall\ i=1...d}$
  avec D_i la i^{ieme} donnée de l'algorithme et d le nombre de données manipulées par l'algorithme.

- Soit Mem l'ensemble des mémoires nécessaires pour le stockage des données :
  $Mem = \{M_i\}_{\forall\ i=1...m}$
  avec M_i la i^{ieme} mémoire et m le nombre de mémoires nécessaires pour le stockage des données. Rappelons que m est une variable connue du système, et que son calcul, effectué lors la deuxième étape de notre méthodologie tient compte des conflits d'accès des données.
- Soit Act l'ensemble des acteurs du système (initiateurs de lectures et/ou d'écritures dans les mémoires) :
  $Act = \{A_i\}_{\forall\ i=1...a}$ avec A_i le i^{ieme} acteur et a le nombre d'acteurs du système.
- Soit P_{ij} la variable indiquant si la i^{ieme} donnée est stockée dans la j^{ieme} mémoire :

$$Pij = \begin{cases} 1 & \text{si } D_i \text{ est stockée dans } M_j \\ 0 & \text{sinon} \end{cases}$$

  Notons que ces éléments sont les inconnues du problème à optimiser.
- Soit W_{ij} la variable indiquant si la i^{ieme} donnée est produite (écrite) par le j^{ieme} acteur du système :

$$Wij = \begin{cases} 1 & \text{si } D_i \text{ est produite par } A_j \\ 0 & \text{sinon} \end{cases}$$

- Soit R_{ij} la variable indiquant si la i^{ieme} donnée est consommée (lue) par le j^{ieme} acteur du système,

$$Rij = \begin{cases} 1 & \text{si } D_i \text{ est consommée par } A_j \\ 0 & \text{sinon} \end{cases}$$

- Soit Tps l'ensemble des temps pour lesquels au moins une opération d'accès à la mémoire est réalisée :
  $Tps = \{T_i\}_{\forall\ i=1...t}$, avec t le nombre de temps d'accès différents.
- Soit TA_{ij} la variable indiquant si la i^{ieme} donnée est consommée ou produite (lue ou écrite = accès) au temps j,

$$TA_{ij} = \begin{cases} 1 & \text{si } D_i \text{ est accédée au temps } T_j \\ 0 & \text{sinon} \end{cases}$$

L'objectif étant de rechercher une solution présentant le minimum de connexions entre les acteurs du système et l'organisation mémoire. La fonction coût à minimiser est donc la suivante :

$$Cout = MIN\left(\sum_{i=1}^{a}\sum_{j=1}^{m} C_{ij}\right) \tag{1}$$

Avec C_{ij} la variable indiquant s'il existe une connexion entre le i^{ieme} acteur du système et la j^{ieme} mémoire :

$$Cij = \begin{cases} 1 & \text{si } A_i \text{ est connecté à } M_j \\ 0 & \text{sinon} \end{cases}$$

Cette minimisation doit être effectuée sous les contraintes que nous énumérons ci-dessous :

- Contrainte 1 : toute donnée doit être stockée dans une mémoire et une seule :

$$\sum_{j=1}^{m} P_{ij} = 1 \qquad \forall\ i = 1 \ldots d \tag{2}$$

  On pourra, par la suite, proposer une extension de la méthode au stockage de la même donnée dans plusieurs bancs mémoires. Ce cas nécessitera une attention toute particulière concernant le problème de cohérence des lectures et écritures des données. Problème similaire au problème classique de cohérence de cache des systèmes multi-processeurs ;
- Contrainte 2 : toute mémoire ne peut avoir plus de 1 accès par temps :

$$MAX_{\forall\ k=1...t}\left(\sum_{i=1}^{d} P_{ij} * T_{ik}\right) = 1 \qquad \forall\ j = 1 \ldots m \tag{3}$$

  Le nombre maximum d'accès à une mémoire pour le temps k doit être strictement égal à 1. Une valeur supérieure à 1 indiquerait qu'il existe au moins un temps pour lequel la mémoire est sollicitée pour plus d'un accès. Une valeur inférieure à 1 (égale à zéro en l'occurrence), indiquerait que la mémoire n'est jamais sollicitée, donc qu'elle est inutile.
- Contrainte 3 : si une donnée l produite (respectivement consommée) par un acteur i est stockée dans une mémoire j alors il doit exister une connexion entre l'acteur i et la mémoire j

$$C_{ij} = \sum_{l=1}^{d} ((W_{li}\ \vee\ R_{li}) * P_{lj}) \qquad \forall\ i = 1 \ldots a, \qquad \forall\ j = 1 \ldots m \tag{4}$$

  Pour tout couple (acteur i, mémoire j), il existe une connexion entre i et j si l'acteur i fait une lecture ou une écriture de la donnée l et que cette donnée l est stockée dans la mémoire j (la notation $\vee$ correspond à l'opération *ou logique*).

La minimisation de la fonction de coût permet de déterminer les positions des données en mémoires et donc, de déterminer la connectique minimale. Cette formulation suppose que les mémoires sont équivalentes, ce qui est acceptable pour l'ensemble des mémoires d'un niveau hiérarchique donnée. Ainsi dans le cas d'une hiérarchie mémoire, il suffit de formuler le problème pour chaque niveau pour trouver le nombre minimal d'interconnexions à mettre en œuvre. Cette formulation suppose aussi que les mémoires ne disposent que d'un seul port d'accès. Une extension pourrait être envisagée si les mémoires disponibles disposent de plusieurs ports de lecture/écriture. Dans ce cas, la contrainte 2 (voir équation 3) imposée de façon homogène pour

toutes les mémoires, sera revue et exprimée selon les caractéristiques de chaque mémoire.

L'expression du problème est indépendante du temps, ce qui signifie que la solution n'est pas implémentable dans l'état. Si la solution proposée indique qu'un acteur doit être connecté à une ou plusieurs mémoires, encore faut-il savoir à quels instants ces connexions sont à assurer. Une phase d'ordonnancement de ces connexions est alors à réaliser. Cette phase correspond finalement à une phase de définition temporelle de la configuration de la connectique.

4 MISE EN ŒUVRE SUR UNE APPLICATION

Dans cette section, nous appliquons notre méthodologie à une partie d'une application. L'algorithme retenu est la transformée en cosinus discrète (DCT) à deux dimensions. En effet, les standards actuels de compression d'images (JPEG, MPEG, H26x) se basent sur une décomposition en blocs de l'image à traiter et une transformation DCT_{2d} est appliquée sur chaque bloc. Sans reprendre le développement mathématique, et pour une bonne compréhension de l'algorithme rappelons brièvement les formules de calcul de la DCT_{2d} :

$$Y(k,l) = \alpha(k,l) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x(i,j)\, c(i,k)\, c(j,k) \quad (5)$$

$Y(k,l)$ est la DCT_{2d} de l'ensemble des pixels du bloc image source $x(i,j)$, $c(i,j)$ représentent les coefficients cosinus et $\alpha(k,l)$ des facteurs multiplicatifs.
Sous sa forme matricielle, la DCT_{2d} se ramène à des produits de matrices :

$$Y = DCT_{2d}(x) = \underbrace{[C] * [x]}_{ImgTmp} * [C]^T = ImgDst \quad (6)$$

Sous cette forme il est évident qu'une DCT_{2d} sur une image est le résultat de l'application successive de deux DCT_{1d} unidimensionnelles suivant les deux axes x et y. Sur la figure 6, nous donnons une illustration graphique de cette décomposition pour une $DCT_{2d}(4x4)$.

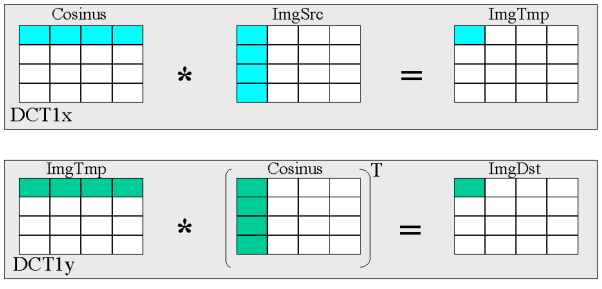

FIG. 6 – Principe de la $DCT_{2d}(4x4)$

Nous proposons de découper cette transformée en deux taches, la première tâche $DCT1_x$ (en haut sur la figure 6) réalise le premier produit de matrices de l'image source avec la matrice des cosinus, la seconde tâche $DCT1_y$ (en bas sur la figure 6) réalise le produit de matrices entre la matrice de cosinus et l'image intermédiaire produite précédemment.
Sur la base de ce découpage en tâches, le fonctionnement se déroule en deux étapes. La première étape réalise une itération de la tâche $DCT1_x$. Une fois cette étape réalisée, la tâche $DCT1_y$ est exécutée et travaille sur le résultat de la première tâche. Il s'en suit alors un séquencement pipeliné des tâches $DCT1_x$ et $DCT1_y$. Si l'on fait abstraction de la première phase (démarrage du pipeline), le déroulement des calculs engendre alors les accès mémoire présentés à la figure 7. L'axe des abscisses représente les temps de cycle du système alors que l'axe des ordonnées représente les adresses mémoires. Pour chaque accès à l'adresse a réalisé au cycle t, un "tiret" est placé à l'abscisse t et l'ordonnée a. Par exemple, au temps de cycle 1, deux ressources vont effectuer des requêtes à la mémoire, l'une demande un pixel de l'image source et un coefficient de la matrice des cosinus, tandis que l'autre demande un pixel de l'image temporaire et le même coefficient de la matrice des cosinus. Finalement, on obtient bien trois requêtes à la mémoire. Notons que cette figure ne différencie pas les opérations de lecture et d'écriture.

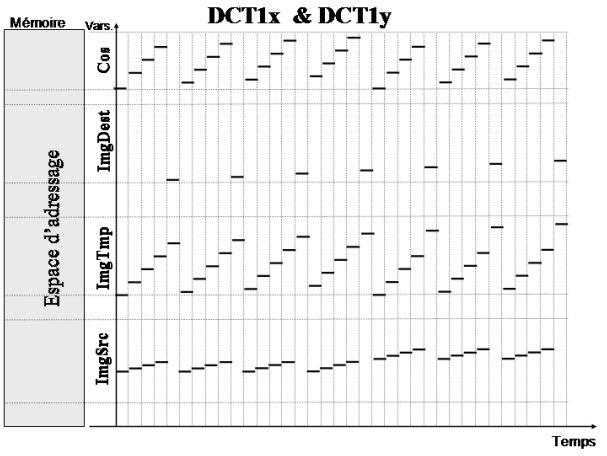

FIG. 7 – Accès mémoire de la $DCT_{2d}(4x4)$. Chaque accès est représenté par un tiret dans le temps de cycle correspondant et à l'adresse correspondant à la donnée attendue

Dans cet exemple, la difficulté apparaît pour les accès qui concernent les matrices $Cosinus$ et $ImgTmp$, puisque ces deux matrices sont accédées simultanément par les deux blocs réalisant les calculs de la DCT_{2d}. Pour la matrice $ImgTmp$ plus particulièrement, le parallélisme des accès conduira l'étape de placement des données à distribuer les colonnes (ou les lignes) de la matrice dans des mémoires différentes (comme indiqué dans la figure 8).
L'architecture proposée est celle qui devrait être fournie par la résolution du problème d'interconnexion. Notons que l'outillage informatique complet de cette proposition est en cours.

Le rôle du module interface consiste à préparer les données pour chaque module de calcul de la DCT. Des interconnexions seront mises en œuvre entre les modules de calcul et les bancs mémoires. Finalement, l'interface construite peut être schématisée par la figure 8. On voit très clairement le partage des matrices $Cosinus$ et $ImgTmp$ apparaître. La logique de contrôle associée à l'interface est adaptée pour différencier ces accès, de façon à aiguiller les données et à cadencer les lectures/écritures vers et depuis la mémoire. Le contrôleur réalisant cette tâche n'est pas détaillé. Il est globalement constitué d'une machine d'états évoluant au rythme des requêtes mémoires et pilotant l'extraction des données ou leur rangement dans les mémoires.

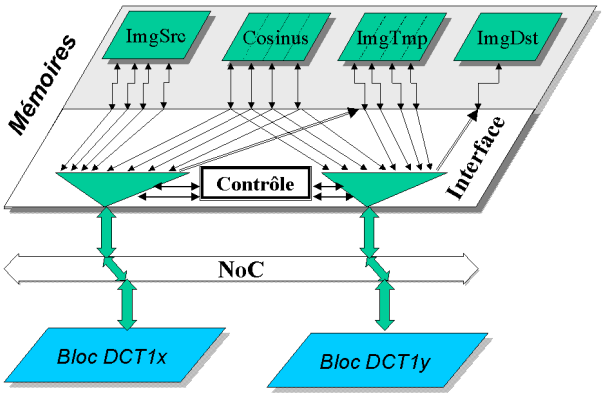

FIG. 8 – Connexion des mémoires aux blocs de calcul

5 CONCLUSION

Dans cet article, nous proposons une approche pour résoudre le problème de l'interconnexion des mémoires avec l'ensemble des acteurs d'un système complet tel que peut l'être un SoC. L'approche que nous proposons s'appuie sur une première étape dont l'objectif est de déterminer le nombre de mémoires nécessaires. Une fois cette étape réalisée, nous proposons de fixer le placement des données dans les mémoires et par conséquent, de mettre en place une interface de connexion entre ces mémoires et le réseau véhiculant l'ensemble des requêtes des acteurs du système. La formulation que nous proposons est basée sur le formalisme ILP et nous permet de trouver le nombre minimal d'interconnexions à gérer par l'interface mémoire. On estime que l'interface est un composant essentiel dans l'architecture mémoire du SoC, du moins pour deux raisons :

– Minimiser le nombre d'interconnexions global, conduit à des meilleures performances en terme de surface et consommation énergétique (pas de transfert de données entre blocs).
– L'interface permet de faire abstraction de l'implémentation matérielle de la mémoire dans le sens ou les autres modules n'ont pas besoin de connaître a priori, ni le nombre de bancs mémoire, ni l'emplacement physique des données. C'est l'interface qui se charge de présenter un espace d'adressage logique selon le modèle de programmation adopté.

Actuellement nous étudions comment cette proposition peut être couplée aux techniques de placement de données afin de satisfaire l'ensemble de la problématique de stockage de données au sein d'un SoC.

BIBLIOGRAPHIE

[Catthoor, 1999] CATTHOOR, F. (1999). Energy delay efficient data storage and transfer architectures and methodologies : Current solutions and remaining problems. *Journal of VLSI Signal Processing Systems*, 21(3):219–231.

[Catthoor *et al.*, 2000] CATTHOOR, F., DUTT, N. D. et KOZYRAKIS, C. E. (2000). How to solve the current memory access and data transfer bottlenecks : at the processor architecture or at the compiler level. *In Proceedings of the conference on Design, automation and test in Europe*, pages 426–435, Paris, France. ACM Press.

[Catthoor *et al.*, 1998] CATTHOOR, F., WUYTACK, S., GREEF, E. D., BALASA, F., NACHTERGAELE, L. et VANDERCAPPELLE, A. (1998). *Custom Memory Management Methodology*. Kluwer Academic Publishers, Norwell, MA, USA.

[Chillet *et al.*, 2005] CHILLET, D., ABDELOUEL, L. et SENTIEYS, O. (2005). Modèle générique de hiérarchie mémoire pour l'exploration architecturale. *In SympA2005 : 8ème édition de SYMPosium en Architectures nouvelles de machines,*, page 12, Le Croisic, France.

[Crummey *et al.*, 2001] CRUMMEY, J. M., WHALLEY, D. et KENNEDY, K. (2001). Improving memory hierarchy performance for irregular applications using data and computation reorderings. *In Int. J. Parallel Program.*, volume 29, pages 217–247, Norwell, MA, USA. Kluwer Academic Publishers.

[Jacob *et al.*, 1996] JACOB, B. L., CHEN, P. M., SILVERMAN, S. R. et MUDGE, T. N. (1996). An analytical model for designing memory hierarchies. *IEEE Transactions on Computers*, 45(10):1180–1194.

[Kavvadias *et al.*, 2001] KAVVADIAS, N., CHATZIGEORGIOU, A., ZERVAS, N. et NIKOLAIDIS, S. (2001). Memory hierarchy exploration for low power architectures in embedded multimedia applications. *In Proceedings of the 2001 IEEE International Conference On Image Processing (ICIP-01)*, pages 326–329.

[Lyuh et Kim, 2004] LYUH, C.-G. et KIM, T. (2004). Memory access scheduling and binding considering energy minimization in multi-bank memory systems. *In DAC '04 : Proceedings of the 41st annual conference on Design automation*, pages 81–86, San Diego, CA, USA. ACM Press.

[Ouaiss et Vemuri, 2001] OUAISS, I. et VEMURI, R. (2001). Hierarchical memory mapping during synthesis in fpga-based reconfigurable computers. *In DATE '01 : Proceedings of the conference on Design, automation and test in Europe*, volume 19, pages 650–657, Munich, Germany. IEEE Press.

[Schmit et Thomas, 1998] SCHMIT, H. et THOMAS, D. E. (1998). Address generation dor memories containing multiple arrays. *In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 17, pages 377–385.

[Schwaderer, 2002] SCHWADERER, W. D. (2002). Solving (soc) shared memory resource challenges. IP Based SoC Design'2002, International Workshop & Exhibition. Sonics Inc. (USA).